An Overview of the Feedforward Neural Network Technique and its Application to SNO Event Classification SNO-STR-95-037

S.J. Brice, Oxford University

15 June 1995

Abstract

This paper presents a general description of pattern classification using a feedforward neural network with particular reference to the use of such a technique for SNO event classification. The separate tasks of event by event and statistical hit pattern recognition are described.

1 Introduction

In the last five years neural network techniques have begun to be applied to high energy physics problems [Den92] [Pet92] and particularly to those involving pattern recognition [BT92] [DC92]. In the context of SNO such techniques are crying out to be used for event classification via hit pattern recognition. This paper presents an introduction to just one neural net paradigm; the feedforward network, which is by far the most popular method and is ideally suited to the questions SNO needs answered. Two kinds of pattern recognition are relevant to the experiment. Event by event classification is needed to identify the nature of individual hit patterns and is useful when analysing supernova events. For dealing with the solar neutrino flux then statistical classification is more relevant and involves taking a sample of hit patterns and finding the fraction that belongs to each event class. The structure of this paper separates into two parts. First the feedforward neural net is described via its application to the event by event problem and then these ideas are modified to handle the statistical classification task.

2 Event by Event Hit Pattern Recognition

2.1 The Task

Given a single PMT hit pattern the task of event classification breaks into two parts:

1. Extract a number of parameters from the hit pattern. These parameters should describe the pattern and capitalise on the intrinsic differences between patterns from the various event classes.

2. Break up the parameter space into fixed regions with each region being assigned to an event class. The parameter values extracted from a hit pattern then specify a point in parameter space and the region within which this point falls then gives the event class.

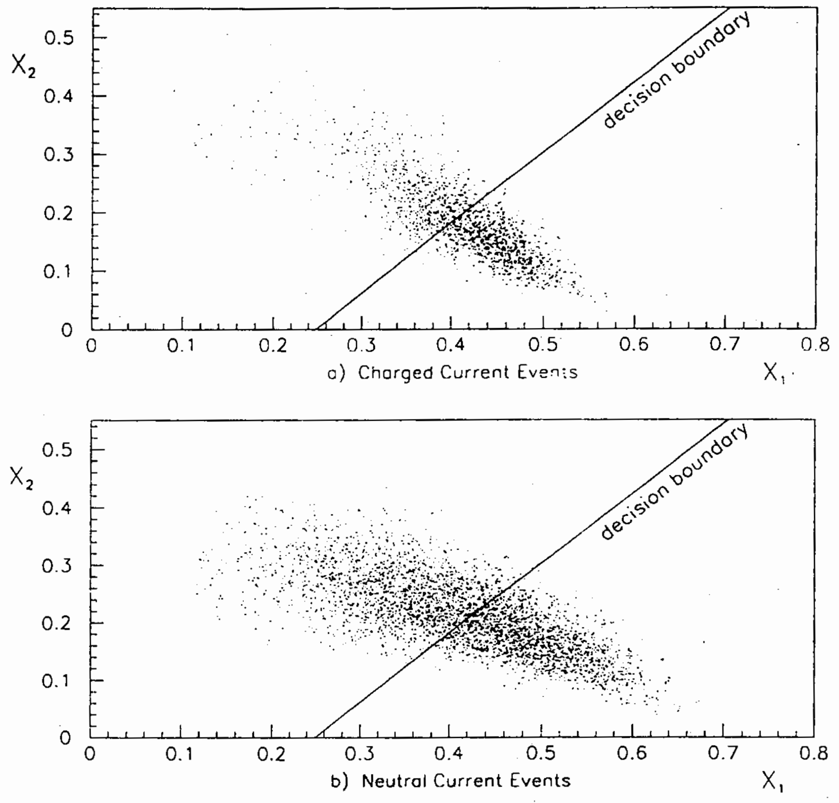

Figure 1: Locations in 2D parameter space of a) 5000 charged current events b) 5000 neutral current events

The procedure outlined above hinges on the appropriate choice of extracted parameters. This choice is far from obvious, but will not be discussed in this paper. The following sections assume the completion of Part 1 of the classification proceedure and go on to describe the way in which a neural network can achieve Part 2.

2.2 The Simplest Case

It is convenient to start with the simplest possible example and then generalise it. Consider the extraction of two parameters from each hit pattern and the classification of hit patterns into two classes; neutral and charged current events. As shown in Figure 1 the two parameters extracted from a hit pattern then describe a point in the 2D parameter space. It can be seen that, for the parameters chosen, neutral and charged current events fall in different but overlapping regions of the space. The best extraction parameters are those for which this overlap is minimised.

The simplest way of dividing the parameter space of Figure 1 into two regions (one for each of the two event classes) is with a linear cut or decision boundary as shown. Events lying above and to the left of the cut are then classified as charged current and those below and to the right as neutral current. It can also be seen that with the rather poor parameters

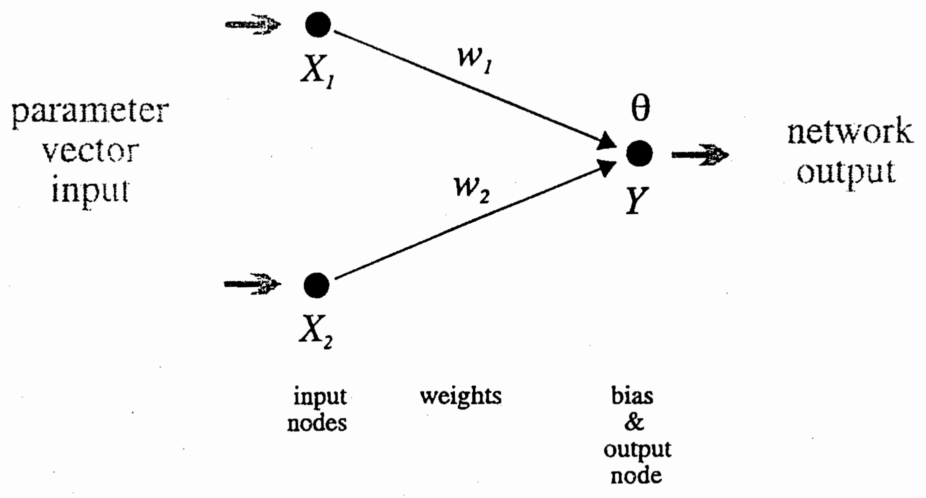

Figure 2: A neural network for the simple 2 parameter/2 class example

chosen a lot of misclassification errors are made in this example. The next two subsections describe how a simple neural network can assign a region/class to the parameters of a hit pattern, and also how the network can optimally place the decision boundary to minimise the misclassification error.

2.2.1 The Operation of a Feedforward Neural Network

A neural network to handle the 2 parameter/2 class example is shown schematically in Figure 2.

The parameters are fed into and held in the input nodes X_i. The weights w_i enable a weighted sum of the inputs to be fed into the output node Y. A bias θ is subtracted from the sum and a non-linear sigmoid function (see Figure 3) of this result is calculated by the output node to produce the output of the network. The network is then carrying out the function:

$$Y(\underline{\mathbf{X}}) = \left[1 + \exp - \left(\sum_i w_i X_i - \theta\right)\right]^{-1} \tag{1}$$

where $\underline{\mathbf{X}}$ is the 2D vector of extracted parameters. An output of $Y = 0.5$ then corresponds to

$$\sum_i w_i X_i = \theta$$

i.e. the input parameters specify a point that lies on a line in parameter space defined by w_i and θ. An output of $Y > 0.5$ means that the input parameters lie on one side of this line and $Y < 0.5$ shows that they lie on the other side. Therefore, given a linear decision boundary defined by w_i and θ, the network provides a way of determining the region within which the extracted parameter point lies.

2.2.2 Training a Feedforward Network by Backpropagation

Given a decision boundary in the parameter space the simple network described in the previous subsection is able to assign the hit pattern to one of the two classes based on whether the output of the net is greater or less than 0.5. On its own this is nothing special, but there

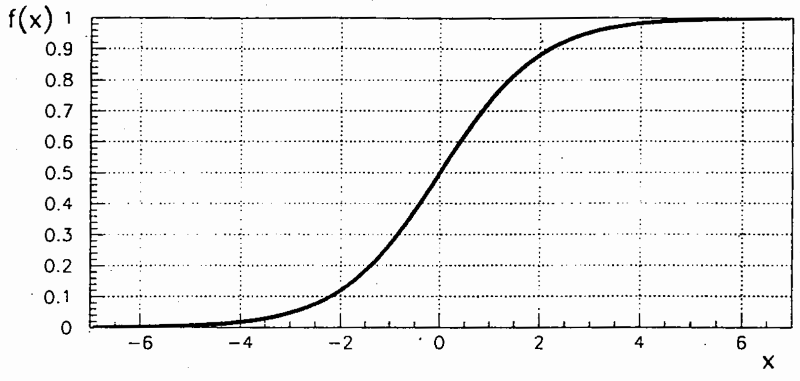

Figure 3: The sigmoid function: $f(x) = [1 + \exp(-x)]^{-1}$

also exists a network training technique, known as backpropagation, that enables the optimal placing of the decision boundary, i.e the optimal setting of w_i and θ.

Training the network to set its internal parameters requires the use of a training set of hit patterns. This set should contain equal numbers of neutral and charged current events to avoid artificially slanting the classification in favour of one class, and the true class of each event must be known (this requires the data to have come from either Monte Carlo or calibration). If a network output of $Y > 0.5$ is to signify a charged current event and $Y < 0.5$ a neutral current then this enables a target network output to be assigned to each hit pattern of the training set:

For true charged current events Target	T = 1.0
For true neutral current events Target	T = 0.0

With p labelling the hit patterns in the training set then X_i^p are the extracted parameters from hit pattern p and T^p is the target output that the network should try to achieve for that pattern. With these designations it is then possible to define a training error E for a particular set of w_i and θ values.

$$E(w_i, \theta) = \frac{1}{2}\sum_p [Y(\underline{\mathbf{X}}^p) - T^p]^2 \tag{2}$$

With a network acheiving perfect classification then the network output $Y(\underline{\mathbf{X}}^p)$ for each hit pattern p will be equal to the target output T^p for that pattern and the training error E will be zero. The more the network outputs differ from their target values then the higher the value of the training error. This provides a criterion for the setting of the internal parameters of the network. The optimal values of w_i and θ (i.e. the optimal positioning of the decision boundary) are those which minimise the training error E. Starting with w_i and θ set randomly between 0 and 1 then they are updated by gradient descent of the error function.

$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i}$$

$$\Delta \theta = -\eta \frac{\partial E}{\partial \theta} \tag{3}$$

The η parameter above is known as the learning rate and controls the size of the update steps (its value is typically about 0.01). Iteration continues until the error function has converged to its minimum value. Although gradient descent is the simplest weight update method it works well in most cases. If convergence is slow or there are problems with local minima of the error function then extensions to gradient descent are available (e.g. the addition of a momentum term). Calculating the update partial derivatives explicitly yields

$$\Delta w_i = -\eta \sum_p [Y(\underline{\mathbf{X}}^p) - T^p][Y(\underline{\mathbf{X}}^p) - 1]Y(\underline{\mathbf{X}}^p)X_i^p$$

$$\Delta \theta = \eta \sum_p [Y(\underline{\mathbf{X}}^p) - T^p][Y(\underline{\mathbf{X}}^p) - 1]Y(\underline{\mathbf{X}}^p) \tag{4}$$

For the simple case under consideration these two subsections have shown how a neural network can optimally set a linear decision boundary in parameter space by altering its internal weights and biases and then with these internal parameters frozen can assign a class to any new input vector presented to it.

2.3 Extending the Simplest Case into Something Useful

What has been described so far is no more than a linear discriminant and there exist a number of equally effective and much simpler statistical techniques for performing the same task (e.g. the Fisher discriminant [Sch92]). The simplest case can easily be extended to handle more than two extracted parameters and an increased number of event classes, but the real advantage of using a feedforward net, and the point at which it leaves the competition behind, comes when it is generalised to form non-linear or curved decision boundaries.

2.3.1 More Inputs and More Outputs

The first generalisation required of the simplest case is to enable the network to deal with more than two extracted parameters. With n parameters then the number of input nodes X_i and the number of weights w_i increases from 2 to n and all the mathematics is identical. This is a trivial extension and simply involves moving from a 2 dimensional parameter space with a single line cut to an n dimensional space with a single hyperplanar cut.

With more than two event classes (e.g. the addition of a background class) then the required extensions are a little more involved but still simple. To distinguish c classes then c output nodes are required (as opposed to 1 node for the 2 class example which is a special case) as is illustrated in Figure refmediumnet. There are now weights w_{ik} connecting every input to every output where i labels the input node and k the output node and each output has its own bias θ_k. The vector of output values Y_k is calculated from the inputs X_i by the logical extension of Equation 1

$$Y_k(\underline{\mathbf{X}}) = \left[1 + \exp - \left(\sum_i w_{ik}X_i - \theta_k\right)\right]^{-1} \tag{5}$$

Each output Y_k corresponds to a particular class and an input parameter vector, once fed through the net, is assigned to the class corresponding to the highest output. To see more graphically how this works consider the two parameter simple example, but now with classification into three event classes i.e. a network with two input and three output nodes. For each output the equation $Y_k = 0.5$ once again corresponds to a line in parameter space as is shown in Figure 5. For each output the argument of its sigmoid function

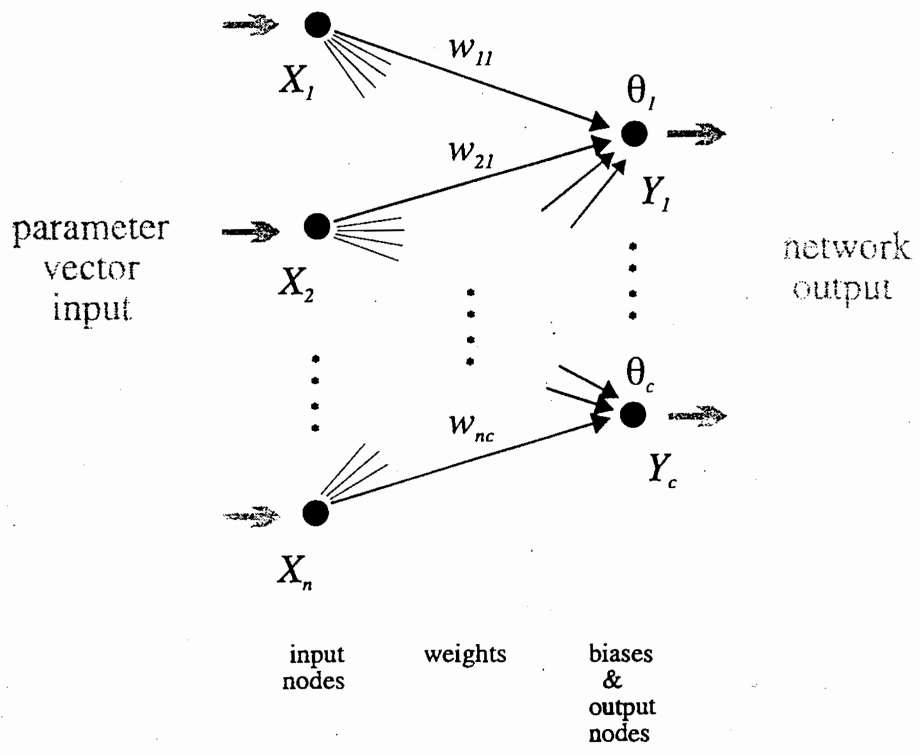

Figure 4: The extension of the simple example to n inputs and c outputs

$$\sum_i w_{ik} X_i - \theta_k$$

is simply the signed distance from this line. Since the sigmoid is a monotonically increasing function then each output value Y_k is a nonlinear measure of the signed distance of the input parameter vector from the line labelled by k. Therefore, by assigning the class of an input vector to that of its highest output the parameter space is broken up into regions. As shown in Figure 5 for the 3 output example each region is labelled by the line from which it is furthest i.e. region k corresponds to output Y_k being highest. It is in this way that the weights w_{ik} and biases θ_k break up the n dimensional parameter space into c regions.

To train such a multioutput network now requires a c dimensional target vector for each hit pattern. This vector has 1 in the slot of the target class and all other entries are zero. An error function is defined as a direct extension of Equation 2 with the output $\underline{\mathbf{Y}}$ and target $\underline{\mathbf{T}}^p$ now being vectors.

$$E(w_{ik}, \theta_k) = \frac{1}{2} \sum_p [\underline{\mathbf{Y}}(\underline{\mathbf{X}}^p) - \underline{\mathbf{T}}^p]^2 \tag{6}$$

The optimal setting of w_{ik} and θ_k then proceeds by gradient descent of this error function as before.

2.3.2 Non-linear Boundaries and Hidden Nodes

Even with the extension to n extracted parameters and c event classes the network described so far is still just a linear discriminant with hyperplanar region boundaries. With a further extension the feedforward network becomes a non-linear classifier with curved region boundaries. This is achieved by inserting a layer of hidden nodes between the input and output

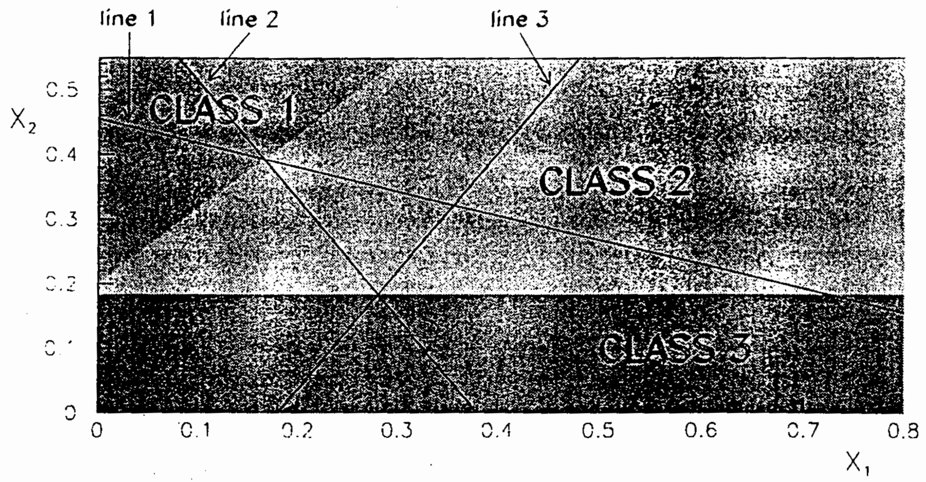

Figure 5: The decision regions for the 2 parameter/3 class extension to the simple example. Each output k has an associated line k given by $\sum_i w_{ik} X_i = \theta_k$. Class k then corresponds to the region which is most distant from line k i.e. its distance is most positive.

nodes as shown in Figure 6. The network has n inputs X_i, h hidden nodes H_j and c outputs Y_k. The input parameters are fed through to the output nodes by the logical extension of the previous examples. Each hidden node calculates a weighted and biased sigmoid of the inputs with weights w_{ij} and biases θ_j and then each output node calculates a weighted and biased sigmoid of the values from the hidden nodes using weights W_{jk} and biases Θ_k. Overall then the network executes the function

$$Y_k(\underline{X}) = \left[1 + \exp - \left(\sum_j W_{jk} \left[1 + \exp - \left(\sum_i w_{ij} X_i - \theta_j\right)\right]^{-1} - \Theta_k\right)\right]^{-1} \tag{7}$$

Alternatively this can be expressed as

$$H_j(\underline{X}) = \left[1 + \exp - \left(\sum_i w_{ij} X_i - \theta_j\right)\right]^{-1}$$

$$Y_k(\underline{X}) = \left[1 + \exp - \left(\sum_j W_{jk} H_j(\underline{X}) - \Theta_k\right)\right]^{-1} \tag{8}$$

With two input parameters the straight lines of Figure 5 now become more general curves with corresponding curved region boundaries. This is entirely due to the use of the sigmoid function which, until now, has been an unnecessary encumbrance. To demonstrate this mathematically is rather messy and not particularly illuminating.

The training of such a multilayer network uses the error function of Equation 6

$$E(w_{ij}, \theta_j, W_{jk}, \Theta_k) = \frac{1}{2} \sum_p [\underline{Y}(\underline{X}^p) - \underline{T}^p]^2 \tag{9}$$

and the weights and biases are updated by the usual gradient descent equations

$$\Delta W_{jk} = -\eta \frac{\partial E}{\partial W_{jk}}$$

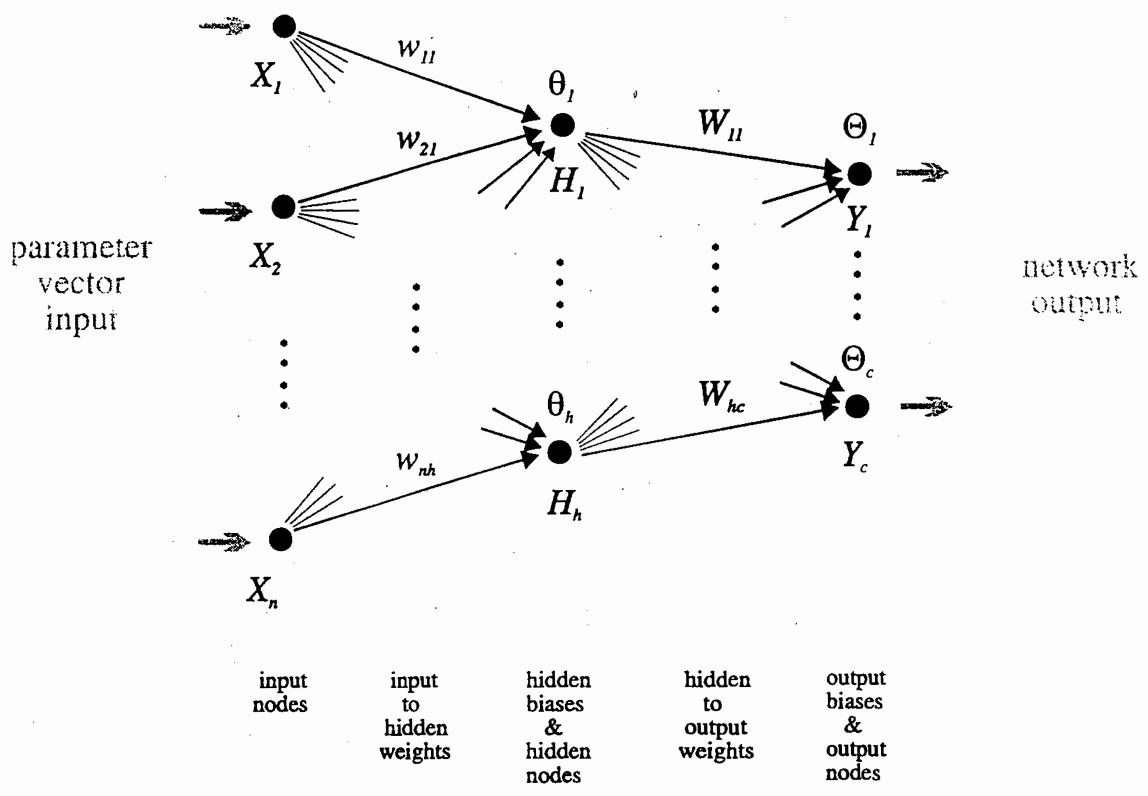

Figure 6: A fully fledged feedforward network with n input, h hidden, and c output nodes

$$\begin{aligned}\Delta\Theta_k &= -\eta\,\frac{\partial E}{\partial \Theta_k}\\ \Delta w_{ij} &= -\eta\,\frac{\partial E}{\partial w_{ij}}\\ \Delta\theta_j &= -\eta\,\frac{\partial E}{\partial \theta_j}\end{aligned} \tag{10}$$

It is instructive to calculate these partial derivatives explicitly. First make the definitions

$$\delta_k^p = Y_k^p[1 - Y_k^p][Y_k^p - T_k^p] \tag{11}$$

$$\delta_j^p = H_j^p[1 - H_j^p]\sum_k W_{jk}\delta_k^p \tag{12}$$

where δ_k^p is known as the error of output node Y_k for pattern p and δ_j^p is the error of hidden node H_j for pattern p. With these assignments then the updates for the weights and biases become

$$\begin{aligned}\Delta W_{jk} &= \eta\sum_p \delta_k^p H_j^p\\ \Delta\Theta_k &= -\eta\sum_p \delta_k^p\\ \Delta w_{ij} &= \eta\sum_p \delta_j^p X_i^p\end{aligned}$$

$$\Delta\theta_j = -\eta \sum_p \delta_j^p \tag{13}$$

These equations have a rather satisfying interpretation. The updates for the hidden to output weights $W_{jk}$are formed by combining a hidden node value H_j^p with an output node error δ_k^p. These output node errors are then weighted and summed to form hidden node errors. The input to hidden weights w_{ij} are then updated by combining an input node value X_i^p with a hidden node error δ_j^p. Overall then network training proceeds by propagating extracted parameter vectors from input to output and then backpropagating errors from output to input in order to update the weights and biases. Hence the algorithm is known as a feedforward neural network with backpropagation.

The extension of the network to include more than one hidden layer is perfectly feasible, but almost certainly unnecessary for SNO purposes. This follows principally from a theorem [HSW90] which states that any functional mapping can be approximated to arbitrary accuracy by a feedforward neural network with just one hidden layer. The proof of this involves demonstrating that the sigmoids of an infinite number of hidden nodes form a complete set of functions. However, second and third hidden layers are used in many applications, as a task can often be accomplished with less hidden nodes overall by using more hidden layers. Since the extraction of parameters from hit patterns constitutes a significant amount of preprocessing, the decision regions required of a network for SNO event classification are likely to be singly connected, simple spaces requiring only one hidden layer to form them.

The inclusion of a hidden layer has introduced a second free parameter into the network algorithm (the learning rate η is the first) - the number of hidden nodes has to be chosen. With too few hidden nodes there is insufficient flexibility to form the necessary decision boundaries and with too many a process known as overtraining can occur, where the network starts to learn the features of individual input patterns rather than the characteristics of the probability distributions from which they are drawn. To understand this overtraining an analogy with curve fitting is useful. Any physicist knows the danger of fitting a set of data points with a curve that has too many adjustable parameters. The result is a curve that fits the points perfectly but does not reproduce the underlying distribution from which the data was taken. Just the same problem occurs with the adjustable weights and biases of the network and the number of training patterns. To avoid overtraining the size of the training set of input vectors must be significantly larger than the number of weights and biases of the network. If W is the number of weights and biases and ϵ is average network error per input pattern (the network error being given by Equation 9 for a set of patterns independent of those used to train the network) then it can be shown [HKP91] that the number of training patterns P should satisfy

$$P > \sim \frac{W}{\epsilon} \tag{14}$$

in order to avoid overtraining.

Whilst there exist some very elegant pruning and growing techniques to kill off or add hidden nodes during the training process, a network for SNO event classification probably has no need of them. This is once again due to the simplicity of the required decision boundaries and the limited number of hidden nodes required to form them. Typically the task of SNO event classification requires a number of hidden nodes that is of the same order as the number of input nodes and often considerably less. So long as the inequality of Equation 14 is satisfied, the classification ability of the network will not be compromised by having more hidden nodes than is strictly necessary.

		Network Assignment	
		CC	NC
True Class	CC	3684	1316
	NC	1274	3726

Figure 7: Example of a confusion matrix for a testing set consisting of 5000 charged current and 5000 neutral current events

2.4 Calibrating the Network and Assigning Errors

2.4.1 Calibration

Once the network has been trained, it should be tested using a labelled data set independent of that used for training. The results of this testing are best displayed using the charmingly named *confusion matrix.* An example of this for a testing set comprising 5000 charged current and 5000 neutral current events is shown in Figure 7. The rows of the matrix are labelled by the true class of the events, and the columns by the network assignment. The matrix entry (i, j) is the number of events that belong to class i and which the network assigned to class j. The goal of perfect event by event hit pattern classification is then a diagonal confusion matrix.

Probably the single most useful number that can be extracted from the confusion matrix is an estimate of the fraction of class assignments that the network gets right. This is known as the overall network purity. If p is the true probability that the network assigns a randomly selected pattern correctly, then the number of correct assignments C from a testing set of N patterns has a binomial probability distribution

$$P(C) = \frac{N!}{C!(N-C)!}\, p^C (i-p)^{N-C} \tag{15}$$

The expectation value of C is then Np and so $\frac{C}{N}$ is an estimate of the network purity where C is given by the trace of the confusion matrix. Since the variance of the binomial is

$$\sigma^2 = Np(1-p)$$

it is possible to put an error on the purity estimate by using the unbiased estimate of the width of the distribution. Bringing all this together. the fraction of class assignments that the network makes correctly can be estimated as

$$\text{Network purity} = \text{Fraction correct} = \frac{C}{N} \pm \left[\frac{1}{N-1}\frac{C}{N}\left(1-\frac{C}{N}\right)\right]^{\frac{1}{2}} \tag{16}$$

where the 1σ error is understood to be the width of a binomial rather than the usual Gaussian width. For the testing results of Figure 7 the network purity is

$$(74.1 \pm 0.4)\%$$

This formalism can obviously be extended to other relevant estimates (e.g. the fraction of the time that the network assigns a hit pattern to class 1 correctly).

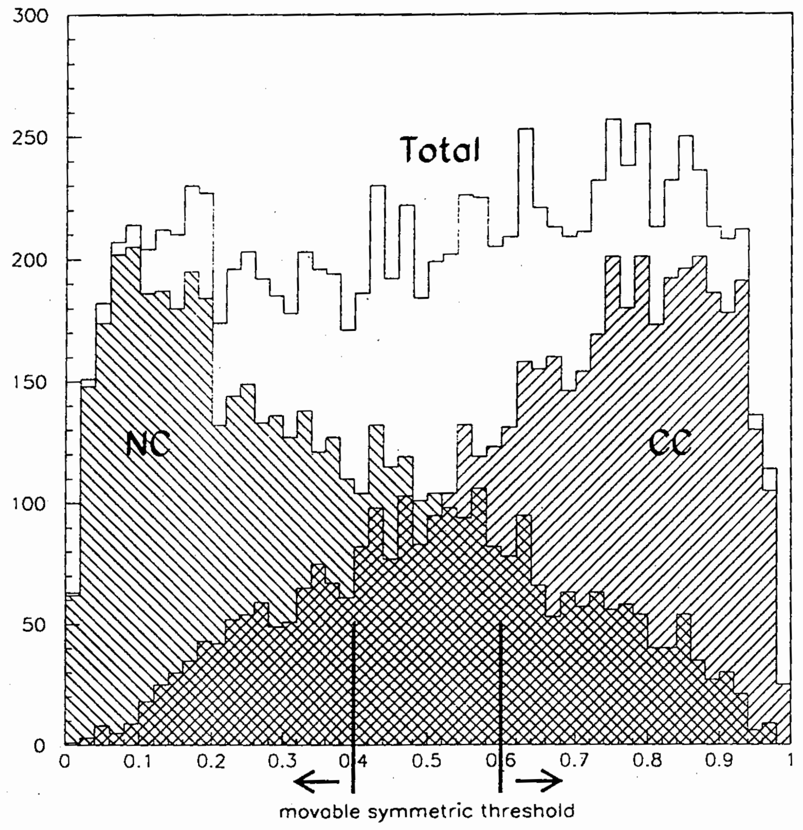

Figure 8: A histogram of the testing set output values of a network with one output node trained to distinguish between neutral (NC) and charged (CC) current events. The two hatched histograms show the output values for the 5000 true NC and 5000 true CC events. The sum of these two histograms is also shown.

Figure 8 shows a histogram of the network output values for the same testing set used to produce the confusion matrix of Figure 7. The network has a single output and has been trained to distinguish between neutral current (NC) and charged current (CC) events. The testing set consists of 5000 NC and 5000 CC events. Events with an output value greater than 0.5 are assigned to the CC class and those with an output less than 0.5 to the NC class and in this way the confusion matrix of Figure 7 is produced.

As will be shown in the next section, the numerical value of an output for a particular input pattern is a measure of the network's confidence in the class assignment it has made. For the single output network of Figure 8, output values around 0.5 indicate that the network is not confident about the class assignment, and confidence increases as the output value tends to zero or one. For a multioutput network, the closer a particular output value is to unity, the more confident is the class assignment. This provides a method of increasing network purity at the expense of overall efficiency. A multioutput network assigns an input pattern to the class corresponding to the highest output value, but if this value fails to exceed some threshold then the event can be thrown out as being unclassifiable. Similarly, a pair of thresholds for

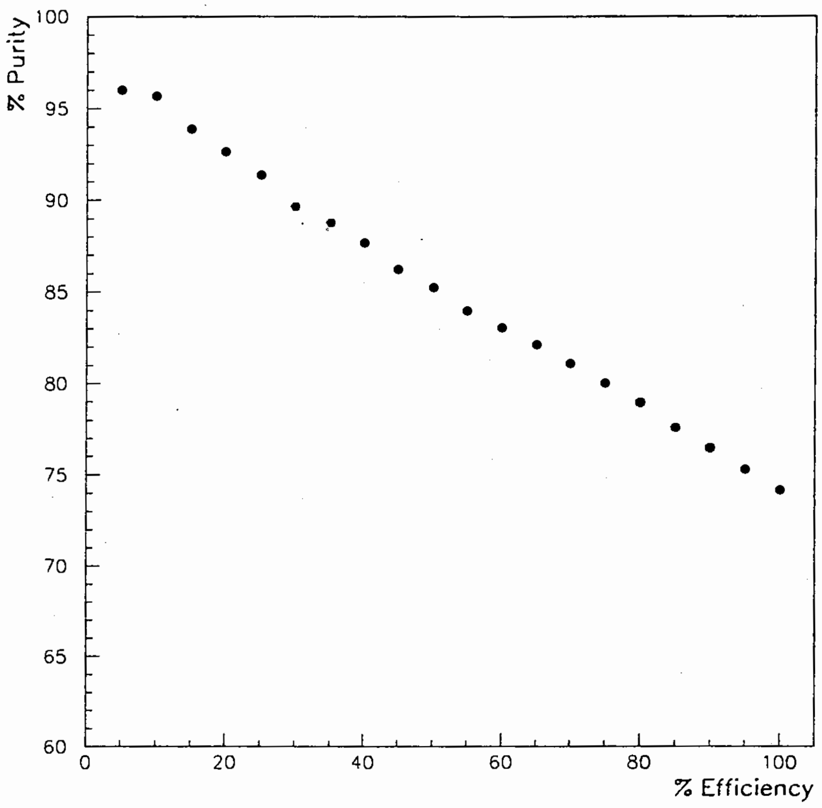

Figure 9: The variation of network purity with overall efficiency for the testing set of outputs from Figure 8 as the symmetric threshold is moved. The error bars vary from 0.4% to 1% and are not shown.

a single output network can be moved out symmetrically from 0.5 as shown in Figure 8 so that events falling between the two thresholds are discarded. The efficiency of a network at a particular threshold setting is simply the fraction of events which are not discarded. As the threshold becomes more stringent and efficiency drops then the network purity will increase. This purity can still be calculated from Equation 16 with the appropriate values for C and N. Figure 9 shows how network purity varies with efficiency as the symmetric threshold is moved for the example outputs of Figure 8.

Unless the output histograms for true NC and true CC events (see Figure 8) are symmetric about 0.5, the thresholding proceedure just outlined for the single output network will result in a biasing of the classification. This bias can be determined from the testing set and thus accounted for.

2.4.2 Assigning Errors

The preceeding sections have shown how a neural network can achieve the task of event by event hit pattern classification once a set of extracted parameters have been decided upon, and how the performance of the network can be calibrated. Nothing has yet been said about the assignment of errors to the classification. It is not clear what an error assignment to a series of event by event classifications means or how it might be interpreted, but such an error specification could be based on either the network purity obtained from the testing set or by using the following theorem: for a network trained on an infinitely large pattern set then the output $Y_k(\underline{X}^p)$ for a particular pattern is the probability of that pattern belonging to class k [Gis90]. To demonstrate, this consider the network error of the output corresponding to the class k

$$E_k = \frac{1}{2}\sum_p [Y_k(\underline{X}^p) - T^p]^2$$

For an infinitely large training set this error becomes

$$E_k = \frac{1}{2}\int \Big(P(\text{class } k|\underline{X})\,[Y_k(\underline{X}) - 1]^2 + P(\overline{\text{class } k}|\underline{X})\,[Y_k(\underline{X})]^2 \Big)\, d\underline{X}$$

where $P(\text{class } k|\underline{X})$ is the probability of the input $\underline{X}$ belonging to class k and $P(\overline{\text{class } k}|\underline{X})$ is the probability that it doesn't. A necessary and sufficient condition for a trained network is that the derivative of this error with respect to the output Y_k be zero. Therefore

$$\int \Big(P(\text{class } k|\underline{X})\,[Y_k(\underline{X}) - 1] + P(\overline{\text{class } k}|\underline{X})\,Y_k(\underline{X}) \Big)\, d\underline{X} = 0$$

With a more rigorous derivation it can be shown that this equation requires that the integrand be zero and so

$$Y_k(\underline{X}) = \frac{P(\text{class } k|\underline{X})}{P(\text{class } k|\underline{X}) + P(\overline{\text{class } k}|\underline{X})}$$

Since the sum of the two conditional probabilities must be unity then

$$Y_k(\underline{X}) = P(\text{class } k|\underline{X}) \qquad (17)$$

This shows that the actual numerical value of the output corresponding to class k is the probability of the input vector belonging to class k. In the parlance of Bayesian statistics the network is calculating an *a posteriori* probability for each class, and as such is behaving as an optimal Bayesian discriminant [DH73] . Although this has been demonstrated for an infinitely large training set, it can be shown [HP90] that as the size of a training set increases then convergence to Equation 17 is rapid and the result is essentially true for even a moderately large number of training patterns. If a proceedure for assigning errors to a set of event by event classifications can be found then either the result of Equation 17 or the calibrated network purity should form its basis.

3 Statistical Hit Pattern Recognition

The preceeding sections have shown how a feedforward neural network can achieve event by event hit pattern classification. Whilst this is useful for supernova events it is not the prefered method of analysing solar neutrino data. The next few sections will demonstrate such a method.

3.1 The Task

Given a set of hit patterns, the task is to ascertain the fraction of the set which belongs to each event class. Once again this is achieved in two parts:

1. Extract a number of parameters from each hit pattern in just the same way as for event by event classification.
2. Feed the extracted parameters through a trained network and fit the distribution of output values to the output distributions obtained from testing. Hence find the fraction of the data set belonging to each event class.

Once again it will be assumed that Part 1 has been achieved and the next section will go on to describe Part 2.

3.2 The Method

The network topology and training for statistical pattern recognition is identical to that already described for event by event recognition. It is convenient, however, to interpret the resulting algorithm in a somewhat different way. Rather than viewing the network as making a decision on each input pattern based on the network output, it is better to think of the network as carrying out a transformation of the extracted parameters from the n dimensional input space to an c dimensional output space. The training process then maximises the separation in output space of the probability distributions for each event class. The network outputs can then be viewed simply as c extracted parameters which the network has optimally chosen given the hit pattern information that is presented to it.

Once a network has been trained, its calibration proceeds with an independent testing set as before. The outputs of each testing pattern are then normalised so that they sum to unity. This does not apply to a single output network, but is necessitated generally by the probabilistic interpretation of the outputs described in Section 2.4.2(this interpretation means that the network output sum will already be close to unity before normalisation). With c as the number of output nodes the output values now lie within an $c-1$ dimensional subspace. This subspace is binned and the number of events belonging to class k falling into the bin labelled by $\tilde{\underline{\mathbf{Y}}}$ is used to produce the distribution $R(\tilde{\underline{\mathbf{Y}}}|\text{class}\,k)$ for each event class. Once normalised these distributions are estimates of the true underlying probability densities $p(\underline{\mathbf{Y}}|\text{class}\,k)$ of the outputs for each class.

Armed with these $R(\tilde{\underline{\mathbf{Y}}}|\text{class}\,k)$ distributions a real data set can then be analysed. The real data is fed through the network, the outputs for each hit pattern are normalised to unity, and the resulting output distribution is binned in the same way as the testing set to produce the real data distribution $D(\tilde{\underline{\mathbf{Y}}})$. With N events in the real data set then the fraction α_k of events belonging to class k can be estimated by a χ^2 fit to the form

$$D(\tilde{\underline{\mathbf{Y}}}) = N\sum_k \alpha_k R(\tilde{\underline{\mathbf{Y}}}|\text{class}\,k) \tag{18}$$

with the constraint

$$\sum_k \alpha_k = 1$$

The errors which should be used in the χ^2 fit are the theoretical statistical uncertainties in the $R(\tilde{\underline{\mathbf{Y}}}|\text{class}\,k)$ distributions. With the reasonable assumption of Gaussian deviations of $R(\tilde{\underline{\mathbf{Y}}}|\text{class}\,k)$ from $p(\underline{\mathbf{Y}}|\text{class}\,k)$ then the error of the bin labelled by $\tilde{\underline{\mathbf{Y}}}$ is

$$\sigma^2(\tilde{\underline{\mathbf{Y}}}) = \sum_k R(\tilde{\underline{\mathbf{Y}}}|\text{class}\,k) \tag{19}$$

The statistical errors on the fraction estimates α_k are obtained from the inverse error matrix in the normal way for a χ^2 fit.

4 Further Considerations

Sections 2 and 3 have shown how event by event and statistical hit pattern recognition can be realised by a feed forward neural network technique. So far all error estimates have been statistical. The next two sections describe how systematic errors may be assigned and then how calibration data might be used to give confidence in the algorithm.

4.1 Systematic Errors

For a network trained on Monte Carlo data, systematic errors will almost certainly dominate those from statistical sources. In event by event classification these systematics influence the purity estimate as well as the relevance of the Bayesian *a posteriori* result of Section 2.4.2. In statistical classification the $R(\underline{\tilde{Y}}|\text{class}\, k)$ distributions are affected [DC92]. A SNO Monte Carlo contains a number of semi-free parameters, either artificial parameters like ESTEPE and AE in the EGS4 code or quantities like the photon scattering length in water and acrylic absorptivity that come from the physics. By varying these semi-free parameters within reasonable bounds and producing as a result a number of testing sets with different parameter settings, the variation of the purity and $R(\underline{\tilde{Y}}|\text{class}\, k)$ distributions can be assessed and systematic errors assigned. It is likely that an independent systematic error can be assigned to the variation of each semi-free parameter, but correlations may need to be catered for. This method for calculating systematic errors for a Monte Carlo trained network is far from ideal, but probably the only way that it can be done.

4.2 The Use of Calibration Data

Confidence in a neural net event analysis will hinge on the use of calibration data. This data can principally be used to calibrate the Monte Carlo. It will probably not be possible to produce a realistic charged current data set, as the fine hit pattern details that the network picks up will not be reproduced by cobbling together a charged current NHIT spectrum from a beta decay source. However, real neutral current data can easily be produced and together with the various other sources can be used to calibrate the Monte Carlo which can subsequently be used to produce appropriate training and testing pattern sets for the network. A second, and more convincing, check on the neural network method involves training the net on Monte Carlo data sets that can be reproduced by calibration sources (e.g. a neutral current and a beta decay data set). The network is then tested on both Monte Carlo and calibration data and the agreement (or otherwise!) of the two testing results gives information on both the accuracy of the Monte Carlo and the trustworthiness of the network.

5 Conclusion

This paper has detailed how the feedforward neural network technique can be used for both event by event and statistical hit pattern classification. Methods for statistical and systematic error analysis have been described as well as an indication of how calibration data might be used. The algorithm itself is well understood, but more work needs to be done clarifying the error analysis and, most importantly, finding the best set of parameters to extract from the hits patterns.

An excellent neural net simulator known as SNNS [Zel95] and produced by the University of Stuttgart is available free via anonymous FTP from `ftp.informatik.uni-stuttgart.de`. It handles feedforward networks as well as many other neural algorithms and has a particularly good X-Window interface.

References

[BT92] Wayne S. Babbage and Lee F. Thompson. The use of neural networks in γ-π^0 discrimination. *Nuclear Instruments and Methods in Physics Research A*, 330:482–486, 1992.

[DC92] The DELPHI Collaboration. Classification of the hadronic decays of the Z^0 into b and c quark pairs using a neural network. *Physics Letters B*, 295:383–395, 1992.

[Den92] Bruce Denby. Tutorial on neural network applications in high energy physics: A 1992 perspective. In *Proceedings of the Second International Workshop on Software Engineering, Artificial Intelligence, and Expert Systems for High Energy and Nuclear Physics*, pages 287–325, La Londe les Maures, France, 1992.

[DH73] Richard O. Duda and Peter E. Hart. *Pattern Classification and Scene Analysis.* Wiley-Interscience, 1973.

[Gis90] H. Gish. A probabilistic approach to the understanding and training of neural network classifiers. In *Proceedings of the 1990 IEEE International Conference on Acoustics, Speech, and Signal Processing*, volume 3, pages 1361–1364, April 1990.

[HKP91] John Hertz, Anders Krogh, and Richard G. Palmer. *Introduction to the Theory of Neural Computation.* Addison Wesley, 1991.

[HP90] J.B. Hampshire and B. Pearlmutter. Equivalence proofs for multi-layer perceptron classifiers and the Bayesian discriminant function. In Touretzky, Elman, Sejnowski, and Hinton, editors, *Proceedings of the 1990 Connectionist Models Summer School*, San Mateo, California, 1990. Morgan Kaufman.

[HSW90] K. Hornik, M. Stinchcombe, and H. White. Universal approximation of an unknown mapping and its derivatives using multi-layer feedforward networks. *Neural Networks*, 3:551–560, 1990.

[Pet92] Carsten Peterson. Pattern recognition in high energy physics with neural networks. In L. Cifarelli, editor, *QCD at 200TeV*, pages 149–163, 1992.

[Sch92] Robert Schalkoff. *Pattern Recognition; Statistical, Structural, and Neural Approaches.* Wiley, 1992.

[Zel95] Andreas Zell, et al. *SNNS User Manual, Version 4.0.* University of Stuttgart, Institute for Parallel and Distributed High Performance Systems, 1995.